

Metrics for Measuring Change Impacts in AspectJ Software Maintenance and Reuse

Sai Zhang, Haihao Shen, and Jianjun Zhao

School of Software
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240, China
{saizhang, haihaoshen, zhao-jj}@sjtu.edu.cn

Abstract. Software metrics play an important role in software development, project management, and system maintenance tasks. They can be used to indicate the degree of system interdependencies among the components and provide valuable feedbacks for better reusability, maintainability and reliability. During system evolution, software change is an essential operation. When software functionalities are added or removed during maintenance, or when the existing programs are modified to reuse, the assessment of change impact in software systems has become a major concern for both developers and maintenance staffs. However, the current research of change impact analysis for aspect-oriented (AO) software is mainly focused on defining new impact analysis models (or techniques) and evaluating their practicality. Although several AO software coupling frameworks have been proposed, metrics for measuring the change impacts for AO software are still missing. In this paper, we present a change metrics suite for AspectJ software, to measure the impact of changes during system evolution. To evaluate our proposed metrics, we have implemented a change metrics analysis framework, called Cemata. The experiment on seven AspectJ benchmarks suggests that the proposed metrics provide helpful information to evaluate the changes in an AO system.

1 Introduction

The growing costs of software maintenance have been a major concern for developers and users of software systems. A typical software system, continuously evolves to meet ever-changing user needs. As a system evolves, new features are added and existing functionalities are removed or modified. In particular, when we address changes in the current software version, determining whether a change in one part of a program affects the other parts of the program is time-consuming and problematic. Moreover, a common problem with software is that changes to a system, though trivial, may have unexpected, expensive, or even disastrous effects [13]. Software change impact analysis, often called simply *impact analysis*, is a family of approaches for addressing this problem [12].

While impact analysis is often used to evaluate (or predicate) the effects of a change on a system after (or before) that change has been made, a more proactive and quantitative approach is to use software metrics to measure the impact of changes. Software metrics provide a quantitative basis for the validation of models of software development technology and process [8]. The systematic collection

of various metrics during the whole software life cycle will facilitate identifying which practices should be enforced and which ones should be avoided for productivity and quality improvement. For instance, software maintenance staffs may wish to consider several approaches for implementing a requirement change and choose the one that has the lowest estimated impacts according to the defined change metrics. Having a clear quantitative view of the change impacts, programmers can work to a plan rather than simply deal with consequences.

Aspect-Oriented Programming (AOP) [11] has been proposed as a technique for improving separation of crosscutting concerns in software design and implementation. It works by providing explicit mechanisms for capturing the structure of crosscutting concerns in a software system. With the wider adoption of languages such as AspectJ [2], AOP is gaining more popularity and there is growing interest in using aspects to implement crosscutting concerns in object-oriented (OO) systems. When AO features are added to an OO program, or when an existing AO system is modified to reuse, the impact of program modifications needs to be measured to assess. However, AOP languages present unique features and problems for program analysis schemes such as impact analysis and coupling measurement. When performing analysis and measurement on AO software, the aspectual structures such as join point, advice, and aspect, which are different from existing procedural or object-oriented programming languages, must be handled appropriately.

Recently, several change impact analysis techniques [18] [19] [16] and coupling frameworks [4] [5] [20] [15] have been proposed for AO software. However, despite the fundamental importance of change impact assessment during software maintenance and reuse, there are few attempts to define or evaluate the practicability of change metrics for AO software. If an impact analysis or coupling framework is lack of change metrics tool support, it may not reflect the realistic change impacts precisely. As a result, people can not have a quantitative way to measure what effects these changes may have.

In this paper, we present a change metrics suite for AO software. This change metrics suite defines the measurement of change impacts during AspectJ software evolution. To assess the practicability of the metrics suite, we develop a change metrics analysis framework, called Cemata, to calculate these metrics automatically. We perform an empirical study on seven AspectJ benchmarks with 22 versions. The experimental result provides insightful information for the assessment of changes during AO software evolution. To the best of our knowledge, our work is the first attempt to define and evaluate change metrics for measuring change impacts in AO software.

The rest of paper is organized as follows. Section 2 briefly introduces AspectJ and the potential impact of changes in AspectJ software. Section 3 defines a terminology for an AO system. In Section 4, we present a suite of change metrics for AspectJ software. Section 5 describes our tool implementation for the change metrics measurement. An empirical evaluation of the proposed metrics is presented in Section 6. Section 7 discusses some related work. Concluding remarks are given in Section 8.

2 Background

AspectJ [2] is a seamless extension of Java. It provides novel features to implement crosscutting concerns in a software system. A *join point* in AspectJ is a well-defined point in the execution that can be monitored - e.g., a call to a method, method body execution, etc. For a particular join point, the textual part of the program executed during the time span of the join point is called the *shadow* of the join point [11]. Sets of join points may be represented by *pointcuts*, implying that such sets may crosscut the system. Pointcuts can be composed and new pointcut designator can be defined according to these combinations. *Advice* is a method-like mechanism that consists of instructions that execute *before*, *after*, or *around* a pointcut. An *aspect* is a modular unit of crosscutting implementation in AspectJ. Each aspect encapsulates functionality that crosscuts other classes in a program.

AspectJ allows programmers to add new members to classes statically or change the program execution behavior dynamically. For example, the pointcut construct in AspectJ can intercept the program routine and inject new code when certain events occur during its execution. The AspectJ language also provides an *intertype declaration* mechanism to introduce methods, attributes, and interface implementation declarations. Due to the extensive use of subtyping and dynamic dispatch in OO programming language and the inherent semantic intricacies in AO features, when implementing changes in AO software, it involves more complex impacts than in the traditional programming languages:

- **Explicit change impact:** Like in traditional programming languages, the new added, deleted or modified language elements (such as a class, an aspect or a pointcut) can explicitly change the existing program structure. Moreover, in AspectJ programs, because the woven aspect may change control and data dependency to the base code, when either changes in the Java code (like *renaming classes*, *attributes* and *methods*) or the aspect code (like *adding a new pointcut or advice*) occur, the semantics of pointcuts may be silently altered and the program behavior may also be affected.
- **Global change impact:** After a session of source modifications, the non-trivial combination of small changes may affect other parts of program unexpectedly. For example, the *intertype declaration* mechanism in AspectJ can change the system structure dramatically; such as the use of **declare parent** statement, especially when using wildcards, can easily change the global class inheritance tree, without any source editing in a specific class definition.

Concerning the unique features of AspectJ programs, we can define some concrete metrics to measure the specific change impact: (1) the explicit coupling changes for new added, deleted or modified language constructs (methods, attributes and advices. etc), and (2) the global impacts caused by source modifications, such as the changes of inheritance tree depth in the overall system.

3 Terminology

In order to define our change metrics in AO software, we first define a terminology for an AO system.

3.1 System

The AO systems considered in this paper are composed of *aspects* and *classes*. We do not consider *interfaces* specifically because they can be handled similarly with classes.

Definition 1 (AO System) *An AO system S consists of a set of Aspects, $A(S)$ and a set of classes, $C(S)$.*

In a typical AO system, there may exist some inheritance relationships between aspects and/or classes. Here, we focus on the inheritance relationships between aspects and classes.

Definition 2 (Ancestors of an Aspect/a Class) *Let S be an AO system. For each aspect $a \in A(S)$, let $Ancestors(a) \subset A(S)$ be the set of ancestor aspects of a . For each class $c \in C(S)$, let $Ancestors(c) \subset C(S)$ be the set of ancestor classes of c .*

3.2 Modules

In AO software, an aspect can contain several types of module, i.e., advice, intertype declaration, pointcut and method, and a class contains only one type of module called method.

Definition 3 (Modules of an Aspect) *Let S be an AO system. For each aspect $a \in A(S)$, let $\mathcal{A}(a)$ be the set of advices of a , $\mathcal{I}(a)$ be the set of intertype declarations of a , $\mathcal{P}(a)$ be the set of pointcuts of a , and $\mathcal{M}(a)$ be the set of methods of a .*

Definition 4 (Modules of a Class) *Let S be an AO system. For each class $c \in C(S)$, let $\mathcal{M}(c)$ be the set of methods of c .*

3.3 Attributes

In AO software, both aspects and classes can contain attributes (fields) that are either inherited or newly defined. Attributes are formally defined as follows.

Definition 5 (Attributes of Aspects and Classes) *Let S be an AO system. For each $a \in A(S)$, let $Attributes(a)$ be the set of attributes of aspect a . For each $c \in C(S)$, let $Attributes(c)$ be the set of attributes of class c .*

3.4 Changes in AO Software

During the evolution of an AO system, changes can occur in both modules and attributes. To measure the change impacts of an aspect or a class, it is necessary to define the set of changes to both modules and attributes.

Definition 6 (Adding Module Changes) *Let S be an original AO system, and S' be the modified version of S after a session of changes.*

- *The set of new added aspects of S' can be denoted as $\mathcal{AA}(S')$, and the set of new added classes can be denoted as $\mathcal{AC}(S')$.*
- *For each $a \in A(S')$, the set of new added advices of a can be denoted as $\mathcal{AEA}(a)$; the set of new added pointcuts of a can be denoted as $\mathcal{APC}(a)$; the set of new added intertype declarations of a can be denoted as $\mathcal{AID}(a)$ and the set of new added methods of a can be denoted as $\mathcal{AME}(a)$.*
- *For each $c \in C(S')$, the set of new added methods of c can be denoted as $\mathcal{AME}(c)$.*

Definition 7 (Deleting Module Changes) Let S be an original AO system, and S' be the modified version of S after a session of changes.

- The set of deleted aspects and deleted classes from S can be denoted as $\mathcal{DA}(S)$ and $\mathcal{DC}(S)$, respectively.
- For each $a \in A(S)$, the set of deleted advices of a from S can be denoted as $\mathcal{DEA}(a)$; the set of deleted pointcuts of a can be denoted as $\mathcal{DPC}(a)$; the set of deleted intertype declarations of a can be denoted as $\mathcal{DID}(a)$ and the set of deleted methods of a can be denoted as $\mathcal{DME}(a)$.
- For each $c \in C(S)$, the set of deleted methods of c from S can be denoted as $\mathcal{DME}(c)$.

Definition 8 (Modifying Module Changes) Let S be an original AO system, and S' be the modified version of S after a session of changes.

- For each $a \in A(S')$, the set of modified advices, pointcuts, intertype declarations and methods from S to S' can be denoted as $\mathcal{CAB}(a)$, $\mathcal{CPB}(a)$, $\mathcal{CID}(a)$ and $\mathcal{CME}(a)$, respectively.
- For each $c \in C(S')$, the set of modified methods from S to S' can be denoted as $\mathcal{CME}(c)$.

Note that, for each advice (intertype declaration, pointcut or method) in $\mathcal{CAB}(a)$ ($\mathcal{CPB}(a)$, $\mathcal{CID}(a)$, $\mathcal{CME}(a)$ or $\mathcal{CME}(c)$), only the advice (intertype declaration, pointcut or method) body has been modified regardless how many statements are changed.

Definition 9 (Attribute Changes) Let S be an original AO system, and S' be the modified version of S after a session of changes.

- For each $a \in A(S')$, the set of new added, deleted and modified attributes from S to S' are denoted as $\mathcal{AF}(a)$, $\mathcal{DF}(a)$ and $\mathcal{CFI}(a)$, respectively.
- For each $c \in C(S')$, the set of new added, deleted and modified attributes from S to S' are denoted as $\mathcal{AF}(c)$, $\mathcal{DF}(c)$ and $\mathcal{CFI}(c)$, respectively.

Note that, for each attribute $\epsilon \in \mathcal{CFI}(a)$ or $\mathcal{CFI}(c)$, only the initializer part of ϵ has been changed.

4 Change Metrics for AspectJ Software

We next define a change metrics suite in terms of system coupling to measure the change impact of an AO program from various viewpoints. These change metrics can be classified into two categories, namely *explicit impact metrics* (Section 4.1) and *global impact metrics* (Section 4.2).

4.1 Explicit Impact Metrics

The explicit impact metrics can be used to measure the direct impact of *adding*, *deleting* or *modifying* a program element, such as the addition of an aspect, a piece of advice, an intertype declaration, or a method. These metrics can further be classified into three sub-categories, namely *adding change metrics* (Table 1), *deleting change metrics* (Table 2), and *modifying change metrics* (Table 3). Note that the *adding*, *deleting* or *modifying changes* here include both module and attribute changes.

Metrics	Definition
M_1 :	The sum of the longest path length from aspect $a \in \mathcal{AA}(S')$ to the aspect hierarchy root in S' .
M_2 :	The sum of the longest path length from class $c \in \mathcal{AC}(S')$ to the class hierarchy root in S' .
M_3 :	The total number of modules accessing attribute $\epsilon \in \mathcal{AF}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.
M_4 :	The total number of modules accessing attribute $\epsilon \in \mathcal{AF}(c)$ in S' , where class $c \in \mathcal{C}(S')$.
M_5 :	The total number of modules calling method $\mu \in \mathcal{AME}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.
M_6 :	The total number of modules calling method $\mu \in \mathcal{AME}(c)$ in S' , where class $c \in \mathcal{C}(S')$.
M_7 :	The total number of modules affected by pointcut $\rho \in \mathcal{APC}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.
M_8 :	The total number of modules affected by intertype declaration $\iota \in \mathcal{AID}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.
M_9 :	The total number of aspects containing advices possibly triggered by the execution of advice $\alpha \in \mathcal{AEA}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.
M_{10} :	The total number of aspects containing advices possibly triggered by the execution of intertype declaration $\iota \in \mathcal{AID}(a)$ in S' , where aspect $a \in \mathcal{A}(S')$.

Table 1. A catalog of adding change metrics in AO software

Adding change metrics In Table 1, metrics M_1 and M_2 can be used to measure the hierarchy complexity of a new added aspect or class. The deeper a new added aspect or class is in the hierarchy tree, the greater the number of operations it might inherit. Therefore, any modification made in the super classes may affect the behavior of the sub classes. So, it may be more complex to understand the change.

Metrics $M_3 \sim M_6$ can be used to measure the dependencies of a new added class (or aspect) on other modules, in terms of attribute accessing or method calling. For example, M_3 can be formally defined as follows¹:

Let $a \in \mathcal{A}(S')$ be an aspect, $\epsilon \in \mathcal{AF}(a)$ be a new added attribute of a , and $CFA(a, \epsilon)$ be the number of modules accessing the attribute ϵ , in S' . Then we have:

$$\mathcal{M}_3 = \sum_{a \in \mathcal{A}(S')} \sum_{\epsilon \in \mathcal{AF}(a)} CFA(a, \epsilon)$$

Metrics $M_7 \sim M_{10}$ can be used to measure the overall impact of the new added pointcut (intertype declaration or advice) on other modules. For example, M_7 can be formally defined as follows:

Let $a \in \mathcal{A}(S')$ be an aspect, $\rho \in \mathcal{APC}(a)$ be a new added pointcut of a , and $CPA(a, \rho)$ be the number of modules affected by the pointcut ρ , in S' . Then we have:

¹ Due to the space limitation, for the formal definitions of all metrics in Table 1 \sim Table 4, please refer to our technical report [17].

$$\mathcal{M}_7 = \sum_{a \in A(S')} \sum_{\rho \in \mathcal{APC}(a)} CPA(a, \rho)$$

Metrics	Definition
M_{11} :	The sum of the longest path length from aspect $a \in \mathcal{DA}(S)$ to the aspect hierarchy root in S .
M_{12} :	The sum of the longest path length from $c \in \mathcal{DC}(S)$ to the class hierarchy root in S .
M_{13} :	The total number of modules accessing attribute $\epsilon \in \mathcal{DF}(a)$ in S , where aspect $a \in A(S)$.
M_{14} :	The total number of modules accessing attribute $\epsilon \in \mathcal{DF}(c)$ in S , where class $c \in C(S)$.
M_{15} :	The total number of modules calling method $\mu \in \mathcal{DM\mathcal{E}}(a)$ in S , where aspect $a \in A(S)$.
M_{16} :	The total number of modules calling method $\mu \in \mathcal{DM\mathcal{E}}(c)$ in S , where class $c \in C(S)$.
M_{17} :	The total number of modules affected by pointcut $\rho \in \mathcal{DPC}(a)$ in S , where aspect $a \in A(S)$.
M_{18} :	The total number of modules affected by intertype declaration $\iota \in \mathcal{DID}(a)$ in S , where aspect $a \in A(S)$.
M_{19} :	The total number of aspects containing advices possibly triggered by the execution of advice $\alpha \in \mathcal{DEA}(a)$ in S , where aspect $a \in A(S)$.
M_{20} :	The total number of aspects containing advices possibly triggered by the execution of intertype declarations $\iota \in \mathcal{DID}(a)$ in S , where aspect $a \in A(S)$.

Table 2. A catalog of deleting change metrics in AO software

Deleting change metrics In Table 2, metrics M_{11} and M_{12} can be used to measure the direct impact of deleting a class or an aspect in the original system. The deeper a deleted class or an aspect is in the hierarchy, the greater the number of operations it might involve. Therefore, it may cause wider change impact. Metrics $M_{13} \sim M_{16}$ can be used to measure the impact of *deleting changes*, in terms of *attribute accessing* or *method calling*. Metrics $M_{17} \sim M_{20}$ can be used to measure the overall impact of the deleted pointcut (intertype declaration or advice) on other modules. For example, M_{17} can be formally defined as follows:

Let $a \in A(S)$ be an aspect, $\rho \in \mathcal{DPC}(a)$ be a deleted pointcut of a , and $CPA(a, \rho)$ be the number of modules affected by the pointcut ρ , in S . Then we have:

$$\mathcal{M}_{17} = \sum_{a \in A(S)} \sum_{\rho \in \mathcal{DPC}(a)} CPA(a, \rho)$$

Metrics	Definition
M_{21} :	The number changes of modules accessing attribute $\epsilon \in \mathcal{CFI}(a)$ from S to S' , where aspect $a \in A(S')$.
M_{22} :	The number changes of modules accessing attribute $\epsilon \in \mathcal{CFI}(c)$ from S to S' , where class $c \in C(S')$.
M_{23} :	The number changes of modules calling method $\mu \in \mathcal{CM}\mathcal{E}(a)$ from S to S' , where aspect $a \in A(S')$.
M_{24} :	The number changes of modules calling method $\mu \in \mathcal{CM}\mathcal{E}(c)$ from S to S' , where class $c \in C(S')$.
M_{25} :	The number changes of modules affected by pointcut $\rho \in \mathcal{CPB}(a)$ from S to S' , where aspect $a \in A(S')$.
M_{26} :	The number changes of modules affected by intertype declaration $\iota \in \mathcal{CID}(a)$ from S to S' , where aspect $a \in A(S')$.
M_{27} :	The number changes of aspects containing advices possibly triggered by the execution of advice $\alpha \in \mathcal{CAB}(a)$ from S to S' , where aspect $a \in A(S')$.
M_{28} :	The number changes of aspects containing advices possibly triggered by the execution of intertype declaration $\iota \in \mathcal{CID}(a)$, where aspect $a \in A(S')$.

Table 3. A catalog of modifying change metrics in AO software

Modifying change metrics In Table 3, Metrics $M_{21} \sim M_{24}$ measure the number of affected modules caused by the modified attributes (or methods), because the behavior of modules accessing the modified attribute (or calling the modified method) may also be altered. These metrics can be used to measure the impact of *modifying* methods and attributes in an AO system. For example, M_{21} can be formally defined as follows:

Let $a \in A(S')$ be an aspect, $\epsilon \in \mathcal{CFI}(a)$ be a modified attribute, and $CFA(a, \epsilon)$ be the number of modules accessing attribute ϵ , in S , and $CFA'(a, \epsilon)$ be the number of modules accessing attribute ϵ , in S' . Then we have:

$$\mathcal{M}_{21} = \sum_{a \in A(S')} \sum_{\epsilon \in \mathcal{CFI}(a)} |CFA'(a, \epsilon) - CFA(a, \epsilon)|$$

Metrics $M_{25} \sim M_{28}$ measure the number of affected modules by the modified pointcuts (advice or intertype declarations). They can be used to determine the overall impact of the modified pointcut (advice or intertype) on the other modules. For example, M_{25} can be formally defined as follows:

Let $a \in A(S')$ be an aspect, $\rho \in \mathcal{CPB}(a)$ be a modified pointcut, $CPB(a, \rho)$ be the number of modules affected by pointcut ρ in S , and $CPB'(a, \rho)$ be the number of modules affected by pointcut ρ in S' . Then we have:

$$\mathcal{M}_{25} = \sum_{a \in A(S')} \sum_{\rho \in \mathcal{CPB}(a)} |CPB'(a, \rho) - CPB(a, \rho)|$$

4.2 Global Impact Metrics

In AspectJ programs, the local source modifications may also affect other parts of program unexpectedly. We define the global impact metrics in Table 4 to

evaluate the overall change impact in the system level. In Table 4, metrics M_{29} and M_{30} can be used to measure the hierarchy changes of all aspects and classes, excluding the new added or deleted ones. For example, M_{29} can be formally defined as follows:

Let $a \in A(S') \cap A(S)$ be an aspect, $DIT(a)$ be the length of the longest path from a to the aspect hierarchy root in S , and $DIT'(a)$ be the length of the longest path from a to the aspect hierarchy root in S' . Then we have:

$$\mathcal{M}_{29} = \sum_{a \in A(S) \cap A(S')} |DIT'(a) - DIT(a)|$$

Metrics $M_{31} \sim M_{34}$ can be used to measure the number change of modules accessing (calling) the *unchanged*² attributes (methods) in the system. They can be used to measure the implicit global change impact. For example, M_{31} can be formally defined as follows:

Let $c \in C(S') \cap C(S)$ be a class, let $\mu \in \mathcal{M}(c)$ but $\mu \notin \mathcal{AME}(c) \cup \mathcal{DME}(c) \cup \mathcal{CME}(c)$ be a method, and $CMC'(c, \mu)$ be the number of modules calling method μ in S' , and $CMC(c, \mu)$ be the number of modules calling method μ in S . Then we have:

$$\mathcal{M}_{31} = \sum_{c \in C(S') \cap C(S)} \sum_{\mu \notin \mathcal{AME}(c) \cup \mathcal{DME}(c) \cup \mathcal{CME}(c)} |CMC'(c, \mu) - CMC(c, \mu)|$$

Metrics	Definition
M_{29} :	The sum of the longest path length changes of aspect $a \in A(S') \cap A(S)$ to the aspect hierarchy root, between S and S' .
M_{30} :	The sum of the longest path length changes of class $c \in C(S') \cap C(S)$ to the class hierarchy root, between S and S' .
M_{31} :	The number changes of modules calling method $\mu \notin \mathcal{AME}(c) \cup \mathcal{DME}(c) \cup \mathcal{CME}(c)$ from S to S' , where class $c \in C(S') \cap C(S)$.
M_{32} :	The number changes of modules calling method $\mu \notin \mathcal{AME}(a) \cup \mathcal{DME}(a) \cup \mathcal{CME}(a)$ from S to S' , where aspect $a \in A(S') \cap A(S)$.
M_{33} :	The number changes of modules accessing attribute $\epsilon \notin \mathcal{AF}(c) \cup \mathcal{DF}(c) \cup \mathcal{CFI}(c)$ from S to S' , where class $c \in C(S') \cap C(S)$.
M_{34} :	The number changes of modules accessing attribute $\epsilon \notin \mathcal{AF}(a) \cup \mathcal{DF}(a) \cup \mathcal{CFI}(a)$ from S to S' , where aspect $a \in A(S') \cap A(S)$.

Table 4. A catalog of global impact metrics in AO software

4.3 Summary

We have presented a suite of change metrics for AO software. Since all the metrics defined above are absolute metrics, they have the following property in common:

² The "unchanged" attributes (or methods) are the ones which are not new added, deleted or modified during software evolution.

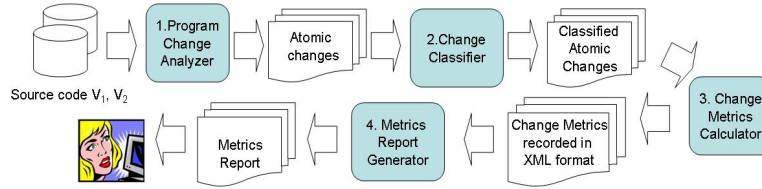


Fig. 1. Overview structure of Cemata framework.

- In general, the larger is the value of a metric, the wider impact the changes have.

The change metrics defined above are classified into several categories, which are amenable for programmers or software maintenance staffs to observe which kind of change metrics has the greatest value, and which type of changes causes the widest change impact.

5 Cemata Framework

To provide automatic tool support for our proposed change metrics, we have implemented Cemata, a change metrics analysis framework for AO software. Cemata is built on top of our existing change impact analysis framework Celadon [18] and coupling metrics tool AJMetrics [15]. It statically analyzes the source code of two versions of an AspectJ program, decomposes their differences into a set of atomic changes, and automatically computes the change metrics values defined in Section 4. The framework consists of four main components: *program change analyzer*, *change classifier*, *change metrics calculator* and *metric report generator*. The overall structure of Cemata is shown in Figure 1.

The program change analyzer is built on top of our Celadon framework [18]. It compares two versions of AspectJ programs, and decomposes their differences into a set of atomic changes [18]. The output atomic changes are fed as inputs of the change classifier. Then, the change classifier module analyzes each of the atomic changes and classifies them into three categories, namely *adding changes*, *deleting changes* and *modifying changes*. These classified change categories are recorded in the XML format, which is convenient for further analysis. The change metrics calculator takes the classified atomic changes generated by change classifier as the inputs, and computes the change metrics defined in Section 4 automatically. Finally, the metrics report generator generates the change metrics analysis report and displays it as a HTML page for developers. Due to the space limitation, please refer to [15] and [18] for more detailed technical issues of Cemata implementation.

6 Experiment

We next present our experiment of applying Cemata to perform change metrics analysis on 22 versions of seven AspectJ benchmarks.

Programs	#Loc	#Ver	#Me	#Shad	Programs	#Loc	#Ver	#Me	#Shad
Quicksort	111	3	18	15	Tracing	1059	4	44	32
Figure	147	4	23	5	NullCheck	2991	4	196	146
Bean	199	3	12	8	Dcm	3423	2	249	359
Spacewar	3053	2	288	369					

Table 5. Subject Programs

6.1 Subject Programs

We use seven AspectJ benchmarks for the experimental study. Table 5 shows the number of lines of code in the original program (#Loc), the number of versions (#Ver), the number of methods (#Me), and the number of shadows (#Shad). These programs are obtained from the AspectJ compiler example package [2] and the abc benchmark package [1]. For each program, we make the first version v_1 a pure Java program by removing all aspectual constructs. To measure the change impact during the software evolution, we take each pair of successive versions of AspectJ benchmark (i.e., v_1 and v_2 , v_2 and v_3 , etc) as the input of Cemata. The result is reported in terms of 34 change metrics defined in Section 4.

6.2 Threats to Validity

Like any empirical evaluation, this study also has limitations which must be considered. In our experience, we have performed change metrics analysis to only subject programs listed in Table 5. Though these subject programs are well-known examples and the last three ones are among the largest programs that we could find, they are smaller than traditional Java software systems. For this case, we can not claim that these experiment results can be necessarily generalized to other programs. On the other hand, threats to internal validity maybe mostly lie with possible errors in our tool implementation and the measure of experiment result. To reduce this kind of threats, we have performed several careful checks. For each result produced by Cemata, we manually inspect the corresponding code to ensure the correctness.

6.3 Result

The result of our experiment is shown in Table 6. Each AspectJ program version is labelled with its version number, such as F3 corresponds to version v_3 of figure. The metrics value between version v_i and v_{i+1} is shown in the v_{i+1} column, such as the change metrics value between Quicksort v_1 and Quicksort v_2 is shown in column Q2.

Metrics $M_1 \sim M_{28}$ in Table 6 are the explicit impact metrics values, while $M_{29} \sim M_{34}$ are the global impact metrics values of the subject programs. The overall change impacts between two benchmark versions are classified in terms of 34 metrics values. From the data, we can see changes addressed in larger benchmarks (T4, N2, N3, N4, D2 and S2) cause more wider impacts than in smaller ones (Q2, Q3, F2 and F3). Besides the class hierarchy change impacts (represented by metrics M_1 , M_2 , M_{11} and M_{12}), the widest impacts are caused by adding new methods (M_6) and pointcuts (M_7). In the experimental result, metrics M_7 and M_9 (also M_{17} and M_{19} , M_{25} and M_{27}) always have the same

Metrics	Q2	Q3	F2	F3	F4	B2	B3	T2	T3	T4	N2	N3	N4	D2	S2
\mathcal{M}_1	1	1	1	0	0	1	1	1	2	5	25	23	26	4	33
\mathcal{M}_2	3	3	6	7	20	2	2	5	4	7	43	42	51	3	53
\mathcal{M}_3	0	0	0	0	0	1	1	1	1	1	0	0	0	1	0
\mathcal{M}_4	0	0	0	3	7	0	0	0	0	0	5	5	5	4	5
\mathcal{M}_5	0	0	0	8	0	2	2	0	0	4	0	0	0	3	7
\mathcal{M}_6	1	1	3	4	9	1	1	2	2	2	26	26	26	1	12
\mathcal{M}_7	2	2	3	0	0	2	2	3	3	3	9	8	14	0	36
\mathcal{M}_8	0	0	0	0	0	7	7	0	0	0	0	0	0	2	0
\mathcal{M}_9	2	2	3	0	0	2	2	3	3	3	9	8	14	0	36
\mathcal{M}_{10}	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
\mathcal{M}_{11}	0	1	0	1	0	0	1	0	1	5	25	26	27	4	24
\mathcal{M}_{12}	3	3	6	6	7	2	2	4	5	7	53	52	52	3	45
\mathcal{M}_{13}	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0
\mathcal{M}_{14}	0	0	0	0	3	0	1	0	0	0	5	5	5	0	3
\mathcal{M}_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
\mathcal{M}_{16}	1	1	3	3	4	0	0	2	2	2	26	26	26	1	10
\mathcal{M}_{17}	0	2	0	3	0	1	1	0	3	3	0	9	8	0	18
\mathcal{M}_{18}	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0
\mathcal{M}_{19}	0	2	0	3	0	0	2	0	3	3	0	9	8	0	18
\mathcal{M}_{20}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
\mathcal{M}_{21}	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
\mathcal{M}_{22}	0	0	0	3	4	1	0	0	0	0	0	0	0	4	2
\mathcal{M}_{23}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
\mathcal{M}_{24}	0	0	0	1	5	0	0	0	0	0	0	0	0	0	2
\mathcal{M}_{25}	2	0	3	3	0	0	0	3	0	0	9	1	6	0	18
\mathcal{M}_{26}	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
\mathcal{M}_{27}	2	0	3	3	0	0	0	3	0	0	9	1	6	0	18
\mathcal{M}_{28}	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
\mathcal{M}_{29}	1	0	1	1	0	1	0	1	1	0	0	3	1	0	9
\mathcal{M}_{30}	0	0	0	1	13	0	0	1	1	0	10	10	1	0	8
\mathcal{M}_{31}	0	0	2	10	28	0	0	5	5	0	10	0	0	0	2
\mathcal{M}_{32}	0	0	0	1	0	16	4	5	0	1	0	0	0	0	72
\mathcal{M}_{33}	0	0	0	3	4	0	0	0	0	0	0	0	0	0	2
\mathcal{M}_{34}	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0

Table 6. Experimental result: the change metrics value between each version pair of benchmarks.

values for all subject program versions. Since in a well-written AspectJ benchmark, pointcut is part of advice definition, therefore, the modules triggered by pointcuts should be the same as the modules affected by advice executions. Otherwise, if M_7 does not equal to M_9 , it may indicate potential defects that since some advices do not match the intended join points.

The global impact metrics seem to be larger than most of the explicit impact metrics in the same column. It reflects that changes in AO software can dramatically affect the program structures (as shown in metrics M_{29} and M_{30}) and dependence relationships (as shown in metrics M_{31} and M_{33}).

6.4 Discussion

In our experiment, the changes between two versions of AO software are represented by a fine-grained collection of change metrics from various viewpoints. From the result, we can get an overview of the change impacts between two versions of AO software during evolution. The metric values give us a quantitative way to assess the changes, and we can see which kind of change metrics has the greatest value, and which type of changes causes the widest change impact. As shown in Table 6, F4 has a much larger value than other versions in the M_2 row. It indicates that there may be dramatic class hierarchy changes caused

by adding new classes³. In column S2, M_{32} is surprisingly high. According to the definition of M_{32} , it reflects that the calling relationship between *unchanged* method in aspect and other modules has been substantially changed⁴. Basically, such information provided by change metrics informs us the *change impacted* parts should be carefully validated.

In Table 6, values in some rows like M_{20} , M_{23} and M_{26} are almost zero. That is because the benchmarks investigated does not contain such specific changes as defined in the metrics definitions. For example, metric M_{20} represents the *the number of aspects whose advices possibly triggered by the deleted intertype declaration*. However, in the AspectJ benchmark package, there is few *intertype declaration deletion* changes. Similarly, metric M_{23} represents the impacts caused by *modifying an existing method in aspect*. However, this change is also not very common between two benchmark versions. For this reason, these cells remain zero for almost all benchmarks. On the other hand, we may need to further evaluate our change metrics on more benchmarks.

The maintenance and reuse phase in software life cycle involve the same subject of assessing impacts of changes made to the system. Another goal of this experiment is to simulate changes during software evolution, and then measure the impacts involving both OO and AO features. We select the version v_1 of each benchmark as a pure Java program, and adding AO features on top v_1 or modifying the existing AO features. We believe our change metrics suite can be used to measure different kinds of change impacts in AspectJ programs and they deserve further investigation.

7 Related Work

We next discuss some related work that has been done in the area of software change metrics, change impact analysis and coupling metrics for AO software.

German and Hindle [9] propose the notion of change metrics and provide a framework for classification of metrics based on the type of change. Lehman et al [14] exemplify the use of metrics to understand the evolution of software. Their metrics are coarse-grained, as they are based on regularly spaced snapshots of the code. There are also many studies [7] [10] used releases and counted their LOCs (or other external attributes of changes) to analyze the changes of a large software system. However, their work does not handle the unique aspectual features and is not intended to measure the change impacts in AO software.

Several change impact analysis techniques [19] [16] [18] have been proposed for AO software. They mainly focus on identifying the impacted source code fragments or tests in the modified software version. In our previous work [18], we present a change impact analysis approach for AspectJ programs and introduce our Celadon framework. In [18], we also identify a catalog of atomic changes for AspectJ programs and propose a change impact analysis model to determine

³ After inspecting the source code of F4, we found five new classes, *figures.NoAspectBoundExpression*, *figures.CFlowStack*, *figures.MyStack*, *figures.CFlowPlusState* and *figures.CFlow* have been new added

⁴ After inspecting the source code of S2 manually, we found a number of 16 methods in aspect *coordinate.Coordinator* have been affected by the changes.

affected program parts, affected tests and their affecting changes. However, a quantitative approach for measuring the change impacts is still missing and the affected program parts are at a coarse method level.

The concept of AO software metrics has also been widely described by researchers. A number of metrics suites and several coupling frameworks have been proposed. Ceccato and Tonella [6] extend OO coupling framework to AO paradigm in terms of the new AO features and propose some new coupling metrics. Zhao [20] also proposes a coupling framework for AO system by analyzing the dependencies between the modules of aspects and classes. Further research on AO coupling framework has also been carried out by Bartsch and Harrison [5] [4] and Bartolomei et al [3]. In our previous work [15], we present a metrics evaluation tool, called AJMetrics, and an evaluation of 16 coupling metrics. However, all the researches mentioned above are focused on measuring the coupling between system modules or the complexity of AO programs, and do not take the change impacts in AO software into consideration.

8 Concluding Remarks

In this paper, we have proposed a change metrics suite for AspectJ programs, to measure the change impacts during software evolution. To assess the practicability of this metrics suite, we also developed a change metrics analysis framework, called Cemata, which analyzes two AspectJ software versions statically and calculates these metrics automatically. The experimental result suggests that the proposed metrics can provide helpful information to evaluate the impact in the system and validate the software changes.

As our future work, we would like to refine our metrics suite and perform further empirical evaluations on larger AO software. We will also study the mathematical properties of the proposed metrics and investigate more applications of this change metrics suite.

Acknowledgements

This work was supported in part by National High Technology Development Program of China (Grant No. 2006AA01Z158), National Natural Science Foundation of China (NSFC) (Grant No. 60673120), and Shanghai Pujiang Program (Grant No. 07pj14058). We would like to thank Cheng Zhang for his valuable comments.

References

1. The AspectBench Compiler. <http://abc.comlab.ox.ac.uk/>.
2. AspectJ Development Tools (AJDT). <http://www.eclipse.org/ajdt/>.
3. T. T. Bartolomei, A. Garcia, C. Sant'Anna, and E. Figueiredo. Towards a unified coupling framework for measuring aspect-oriented programs. In *SOQUA '06: Proceedings of the 3rd international workshop on Software quality assurance*, pages 46–53, New York, NY, USA, 2006. ACM.
4. M. Bartsch and R. Harrison. A coupling framework for AspectJ - extended abstract. In *Evaluation and Assessment in Software Engineering (EASE)*, Keele, UK, 2006.

5. M. Bartsch and R. Harrison. Towards an empirical validation of aspect-oriented coupling metrics. In *ASAT Workshop at the Aspect-Oriented Software Development Conference (AOSD)*, Vancouver, BC, 2007.
6. M. Ceccato and P. Tonella. Measuring the effects of software aspectization. In *Proceedings of the 1st Workshop on Aspect Reverse Engineering (WARE 2004)*, Delft, The Netherlands, Nov. 2004.
7. S. Demeyer, S. Ducasse, and O. Nierstrasz. Finding refactorings via change metrics. In *OOPSLA '00: Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 166–177, New York, NY, USA, 2000. ACM.
8. W. Frakes and C. Terry. Software reuse: Metrics and models. *ACM Computing Surveys*, 28(2):415–435, 1996.
9. D. M. German and A. Hindle. Measuring fine-grained change in software: Towards modification-aware change metrics. In *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05)*, page 28, Washington, DC, USA, 2005. IEEE Computer Society.
10. M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *ICSM*, pages 131–142, 2000.
11. G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proc. 11th European Conference on Object-Oriented Programming*, pages 220–242. 1997.
12. J. Law and G. Rothermel. Incremental dynamic impact analysis for evolving software systems. In *In Proceedings of the International Symposium on Software Reliability Engineering, Nov. 2003.*, page 430, Washington, DC, USA, 2003. IEEE Computer Society.
13. J. L. Lions. Ariane 5, flight 501 failure, report by the inquiry board. European Space Agency, July, 1996.
14. P. W. D. P. M.M. Lehman, J.F. Ramil. Metrics and laws of software evolution-the nineties view. In *In Proc. Metrics 97, Nov. 5-7th, 1997, Albuquerque, NM*, 1997.
15. H. Shen and J. Zhao. An evaluation of coupling metrics for aspect-oriented software. Number SJTU-CSE-07-04, 2007.
16. H. Shinomi and T. Tamai. Impact analysis of weaving in aspect-oriented programming. In *Proc. 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 657–660, Washington, DC, USA, 2005.
17. S. Zhang, H. Shen, and J. Zhao. Metrics for measuring change impacts in AspectJ software maintenance and reuse. Number SJTU-CSE-07-09, 2007.
18. S. Zhang and J. Zhao. Change impact analysis for AspectJ programs. Technical Report SJTU-CSE-TR-07-01, Center for Software Engineering, SJTU, Jan 2007.
19. J. Zhao. Change impact analysis for aspect-oriented software evolution. In *Proc. 5th International Workshop on Principles of Software Evolution*, pages 108–112, May 2002.
20. J. Zhao. Measuring coupling in aspect-oriented systems. In *International Software Metrics Symposium (Metrics2004), (Late Breaking Paper)*, Chicago, USA, 2004.